Indirect Jumps Improve Instruction Sequence Performance

J.A. Bergstra and C.A. Middelburg

Informatics Institute, Faculty of Science, University of Amsterdam,
Science Park 107, 1098 XG Amsterdam, the Netherlands
J.A.Bergstra@uva.nl,C.A.Middelburg@uva.nl

Abstract. Instruction sequences with direct and indirect jump instructions are as expressive as instruction sequences with direct jump instructions only. We show that, in the case where the number of instructions is not bounded, there exist instruction sequences of the former kind from which elimination of indirect jump instructions is possible without a super-linear increase of their maximal internal delay on execution only at the cost of a super-linear increase of their length.

Keywords: instruction sequence performance, indirect jump instruction, maximal internal delay, projectionism.

1998 ACM Computing Classification: D.3.3, F.1.1, F.3.3.

1 Introduction

We take the view that sequential programs are in essence sequences of instructions. Although instruction sequences with direct and indirect jump instructions are as expressive as instruction sequences with direct jump instructions only (see [2]), indirect jump instructions are widely used to implement features of high-level programming language such as Java [6] and C# [7]. Therefore, we consider a theoretical understanding of both direct jump instructions and indirect jump instructions highly relevant to programming. In this paper, we show that, in the case where the number of instructions is not bounded, there exist instruction sequences with direct and indirect jump instructions from which elimination of indirect jump instructions is possible without a super-linear increase of their maximal internal delay on execution only at the cost of a super-linear increase of their length.

The work presented in this paper belongs to a line of research whose working hypothesis is that instruction sequence is a central notion of computer science. The object pursued with this line of research is the development of theory from this working hypothesis. In this line of research, program algebra [1] is the setting used for investigating instruction sequences. The starting-point of program algebra is the perception of a program as a single-pass instruction sequence, i.e. a finite or infinite sequence of instructions of which each instruction is executed at most once and can be dropped after it has been executed or jumped over. This perception is simple, appealing, and links up with practice.

The perception of a program as a single-pass instruction sequence forms part of a point of view taken in the line of research to which the work presented in this paper belongs. It is the point of view that:

- any instruction sequence P, and more general any program P, first and for all represents a single-pass instruction sequence as considered in program algebra;
- this single-pass instruction sequence, found by a translation called a projection, represents in a natural and preferred way what is supposed to take place on execution of P;
- program algebra provides the preferred notation for single-pass instruction sequences.

In [4], the name projectionism is coined for this point of view and its main challenges are discussed. The result of this paper is connected with two of the challenges of projectionism identified in that paper: explosion of size and degradation of performance.

The program notation used in this paper to show that indirect jumps improve instruction sequence performance is $\mathrm{PGLB}_{\mathrm{ij}}$. This program notation is a minor variant of $\mathrm{PGLC}_{\mathrm{ij}}$, a program notation with indirect jumps instructions introduced in [2]. Both program notations are close to existing assembly languages and have relative jump instructions. The main difference between them is that $\mathrm{PGLB}_{\mathrm{ij}}$ has an explicit termination instruction and $\mathrm{PGLC}_{\mathrm{ij}}$ has not. This difference makes the former program notation more convenient for the purpose of this paper.

The performance measure use in this paper is the maximal internal delay of an instruction sequence on execution. The maximal internal delay of an instruction sequence on execution is the largest possible delay that can take place between successively executed instructions whose effects are observable externally. Another conceivable performance measure is the largest possible sum of such delays on execution of the instruction sequence. In this paper, we do not consider the latter performance measure because it looks to be less adequate to the interactive performance of instruction sequences.

This paper is organized as follows. First, we give a survey of the program notation $\mathrm{PGLB}_{\mathrm{ij}}$ (Section 2). Next, we introduce the notion of maximal internal delay of a $\mathrm{PGLB}_{\mathrm{ij}}$ program (Section 3). After that, we present the above-mentioned result concerning the elimination of indirect jump instructions (Section 4). Finally, we make some concluding remarks (Section 5).

2 PGLB with Indirect Jumps

In this section, we give a survey of the program notation $\mathrm{PGLB}_{\mathrm{ij}}$. This program notation is a variant of the program notation PGLB, which belongs to a hierarchy of program notations rooted in program algebra (see [1]). PGLB and $\mathrm{PGLB}_{\mathrm{ij}}$ are close to existing assembly languages and have relative jump instructions.

It is assumed that fixed but arbitrary numbers I and N have been given, which are considered the number of registers available and the greatest natural number that can be contained in a register. Moreover, it is also assumed that fixed but arbitrary finite sets $\mathcal{F}$ of *foci* and $\mathcal{M}$ of *methods* have been given.

The set $\mathfrak{A}$ of *basic instructions* is $\{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$. The view is that the execution environment of a PGLB$_{\mathrm{ij}}$ program provides a number of services, that each focus plays the role of a name of a service, that each method plays the role of a command that a service can be requested to process, and that the execution of a basic instruction $f.m$ amounts to making a request to the service named f to process command m. The intuition is that the processing of the command m may modify the state of the service named f and that the service in question will produce T or F at its completion.

PGLB$_{\mathrm{ij}}$ has the following primitive instructions:

- for each $a \in \mathfrak{A}$, a *plain basic instruction* a;
- for each $a \in \mathfrak{A}$, a *positive test instruction* $+a$;
- for each $a \in \mathfrak{A}$, a *negative test instruction* $-a$;
- for each $l \in \mathbb{N}$, a *direct forward jump instruction* $\#l$;
- for each $l \in \mathbb{N}$, a *direct backward jump instruction* $\backslash\#l$;
- for each $i \in [1, I]$ and $n \in [1, N]$, a *register set instruction* $\mathsf{set}{:}i{:}n$;
- for each $i \in [1, I]$, an *indirect forward jump instruction* $\mathsf{i}\#i$;
- for each $i \in [1, I]$, an *indirect backward jump instruction* $\mathsf{i}\backslash\#i$;
- a *termination instruction* !.

PGLB$_{\mathrm{ij}}$ programs have the form $u_1 ; \ldots ; u_k$, where $u_1, \ldots, u_k$ are primitive instructions of PGLB$_{\mathrm{ij}}$.

On execution of a PGLB$_{\mathrm{ij}}$ program, these primitive instructions have the following effects:

- the effect of a positive test instruction $+a$ is that basic instruction a is executed and execution proceeds with the next primitive instruction if T is produced and otherwise the next primitive instruction is skipped and execution proceeds with the primitive instruction following the skipped one – if there is no primitive instructions to proceed with, deadlock occurs;
- the effect of a negative test instruction $-a$ is the same as the effect of $+a$, but with the role of the value produced reversed;
- the effect of a plain basic instruction a is the same as the effect of $+a$, but execution always proceeds as if T is produced;
- the effect of a direct forward jump instruction $\#l$ is that execution proceeds with the l-th next instruction of the program concerned – if l equals 0 or there is no primitive instructions to proceed with, deadlock occurs;
- the effect of a direct backward jump instruction $\backslash\#l$ is that execution proceeds with the l-th previous instruction of the program concerned – if l equals 0 or there is no primitive instructions to proceed with, deadlock occurs;
- the effect of a register set instruction $\mathsf{set}{:}i{:}n$ is that the contents of register i is set to n and execution proceeds with the next primitive instruction – if there is no primitive instructions to proceed with, deadlock occurs;

– the effect of an indirect forward jump instruction $\mathsf{i}\#i$ is the same as the effect of $\#l$, where l is the content of register i;
– the effect of an indirect backward jump instruction $\mathsf{i}\backslash\#i$ is the same as the effect of $\backslash\#l$, where l is the content of register i;
– the effect of the termination instruction ! is that execution terminates.

$\mathrm{PGLB}_{\mathrm{ij}}$ is a minor variant of $\mathrm{PGLC}_{\mathrm{ij}}$, a program notation with indirect jumps instructions introduced in [2]. The differences between $\mathrm{PGLB}_{\mathrm{ij}}$ and $\mathrm{PGLC}_{\mathrm{ij}}$ are the following:

– in those cases where deadlock occurs on execution of $\mathrm{PGLB}_{\mathrm{ij}}$ programs because there is no primitive instructions to proceed with, termination takes place on execution of $\mathrm{PGLC}_{\mathrm{ij}}$ programs;
– the termination instruction ! is not available in $\mathrm{PGLC}_{\mathrm{ij}}$.

The meaning of $\mathrm{PGLC}_{\mathrm{ij}}$ programs is formally described in [2] by means of a mapping of $\mathrm{PGLC}_{\mathrm{ij}}$ programs to closed terms of program algebra. In that way, the behaviour of $\mathrm{PGLC}_{\mathrm{ij}}$ programs on execution is described indirectly: the behaviour of the programs denoted by closed terms of program algebra on execution is formally described in several papers, including [2], using basic thread algebra [1].[1] Because $\mathrm{PGLB}_{\mathrm{ij}}$ is a minor variant of $\mathrm{PGLC}_{\mathrm{ij}}$, we refrain from describing the behaviour of $\mathrm{PGLB}_{\mathrm{ij}}$ programs on execution formally in the current paper.

3 Internal Delays of $\mathrm{PGLB}_{\mathrm{ij}}$ Programs

In this section, we will define the notion of maximal internal delay of a $\mathrm{PGLB}_{\mathrm{ij}}$ program.

It is assumed that a fixed but arbitrary set $\mathfrak{X} \subset \mathfrak{A}$ of *auxiliary basic instructions* has been given. The view is that, in common with the effect of jump instructions, the effect of auxiliary basic instructions is wholly unobservable externally, but contributes to the realization of externally observable behaviour. In [1], examples are given in which auxiliary basic instructions are useful or even indispensable.

The maximal internal delay of a $\mathrm{PGLB}_{\mathrm{ij}}$ program concerns the delays that takes place between successive non-auxiliary basic instructions in runs of the program. Before we define the maximal internal delay of a $\mathrm{PGLB}_{\mathrm{ij}}$ program, we describe what a run of a $\mathrm{PGLB}_{\mathrm{ij}}$ program is.

A run of a $\mathrm{PGLB}_{\mathrm{ij}}$ program P is a succession of primitive instructions that may be encountered in turn on execution of P.

Because we have not formally defined the behaviour of $\mathrm{PGLB}_{\mathrm{ij}}$ programs on execution, we cannot make formally precise what a run of a $\mathrm{PGLB}_{\mathrm{ij}}$ program is. By the detailed informal description of the effects of the primitive instructions

[1] In several early papers, basic thread algebra is presented under the name basic polarized process algebra.

of PGLB$_{\mathrm{ij}}$ on execution of a PGLB$_{\mathrm{ij}}$ program, the description given above is considered sufficiently precise for the purpose of this paper.

The *maximal internal delay* of a PGLB$_{\mathrm{ij}}$ program P, written $\mathit{MID}(P)$, is the largest $n \in \mathbb{N}$ for which there exist a run $u_1 \dots u_k$ of P, an $i \in [1, k]$, and a $j \in [i, k]$ such that $\mathit{ID}(u_i) = 0$ and $\mathit{ID}(u_j) = 0$ and $\mathit{ID}(u_i) + \dots + \mathit{ID}(u_j) = n$, where $\mathit{ID}(u)$ is defined as follows:

$$\begin{array}{llll}
\mathit{ID}(a) = 0 & \text{if } a \notin \mathfrak{X}\ , & \qquad & \mathit{ID}(\#l) = 1\ , \\
\mathit{ID}(a) = 1 & \text{if } a \in \mathfrak{X}\ , & & \mathit{ID}(\backslash\#l) = 1\ , \\
\mathit{ID}(+a) = 0 & \text{if } a \notin \mathfrak{X}\ , & & \mathit{ID}(\mathsf{set}{:}i{:}n) = 1\ , \\
\mathit{ID}(+a) = 1 & \text{if } a \in \mathfrak{X}\ , & & \mathit{ID}(\mathsf{i\#}i) = 2\ , \\
\mathit{ID}(-a) = 0 & \text{if } a \notin \mathfrak{X}\ , & & \mathit{ID}(\mathsf{i}\backslash\#i) = 2\ , \\
\mathit{ID}(-a) = 1 & \text{if } a \in \mathfrak{X}\ , & & \mathit{ID}(!) = 0\ .
\end{array}$$

In [5], an extension of basic thread algebra is proposed which allows for internal delays to be described and analysed. We could formally describe the behaviour of PGLB$_{\mathrm{ij}}$ programs on execution, internal delays included, using this extension of basic thread algebra. The notion of maximal internal delay of a PGLB$_{\mathrm{ij}}$ program has been defined above so as to be justifiable by such a formal description of the behaviour of PGLB$_{\mathrm{ij}}$ programs on execution.

The time that it takes to execute one basic instruction is taken for the time unit in the definition of the maximal internal delay of a PGLB$_{\mathrm{ij}}$ program. By that $\mathit{MID}(P)$ can be looked upon as the number of basic instruction that can be executed during the maximal internal delay of P. As usual, the time that it takes to execute one basic instruction is called a *step*.

4 Indirect Jumps and Instruction Sequence Performance

In this section, we show that indirect jump instructions are needed for instruction sequence performance.

It is assumed that $\mathsf{bool}{:}1, \mathsf{bool}{:}2, \ldots \in \mathcal{F}$ and $\mathsf{set}{:}\mathsf{T}, \mathsf{set}{:}\mathsf{F}, \mathsf{get} \in \mathcal{M}$. The foci $\mathsf{bool}{:}1$, $\mathsf{bool}{:}2$, ... serve as names of services that act as Boolean cells. The methods $\mathsf{set}{:}\mathsf{T}$, $\mathsf{set}{:}\mathsf{F}$, and get are accepted by services that act as Boolean cells and their processing by such a service goes as follows:

- $\mathsf{set}{:}\mathsf{T}$: the contents of the Boolean cell is set to T, and T is produced;
- $\mathsf{set}{:}\mathsf{F}$: the contents of the Boolean cell is set to F, and F is produced;
- get : nothing is changed, but the contents of the Boolean cell is produced.

The notation $;_{i=1}^{n} P_i$, where $P_1 = u_1^1 ; \ldots ; u_{k_1}^1$, ..., $P_n = u_1^n ; \ldots ; u_{k_n}^n$, is used for $u_1^1 ; \ldots ; u_{k_1}^1 ; \ldots ; u_1^n ; \ldots ; u_{k_n}^n$.

Consider the following PGLB$_{\mathrm{ij}}$ program:

$$\begin{array}{l}
;_{i=1}^{2^k}(-\mathsf{bool}{:}1.\mathsf{get} ; \#3 ; \mathsf{set}{:}1{:}2{\cdot}i{+}1 ; \#(2^k{-}i){\cdot}4{+}2) ; ! ; \\
;_{i=1}^{2^k}(-\mathsf{bool}{:}1.\mathsf{get} ; \#3 ; \mathsf{set}{:}2{:}2{\cdot}i{+}1 ; \#(2^k{-}i){\cdot}4{+}2) ; ! ; \\
\mathsf{i\#}1 ; ;_{i=1}^{2^k}(a_i ; \#(2^k{-}i){\cdot}2{+}1) ; \mathsf{i\#}2 ; ;_{i=1}^{2^k}(a'_i ; !)\ .
\end{array}$$

First, the program repeatedly tests the Boolean cell bool:1. If T is not returned for 2^k tests, the program terminates. Otherwise, in case it takes i tests until T is returned, the content of register 1 is set to $2 \cdot i + 1$. If the program has not yet terminated, it once again repeatedly tests the Boolean cell bool:1. If T is not returned for 2^k tests, the program terminates. Otherwise, in case it takes j tests until T is returned, the content of register 2 is set to $2 \cdot j + 1$. If the program has not yet terminated, it performs a_i after an indirect jump and following this a'_j after another indirect jump. After that, the program terminates. The length of the program is $12 \cdot 2^k + 4$ instructions and the maximal internal delay of the program is 4 steps.

The $\mathrm{PGLB_{ij}}$ program presented above will be used in the proof of the result concerning the elimination of indirect jump instructions stated below.

Theorem 1. *Suppose* `proj` *is a projection from the set of* $\mathrm{PGLB_{ij}}$ *programs to the set of* PGLB *programs with the property that the maximal internal delay of each* $\mathrm{PGLB_{ij}}$ *program is increased at most linear. Moreover, suppose that the number of basic instructions is not bounded. Then* `proj` *is a projection with the property that the length of some* $\mathrm{PGLB_{ij}}$ *program is increased more than linear.*

Proof. Let P be the $\mathrm{PGLB_{ij}}$ program presented above. The maximal internal delay of P is increased at most linear by `proj`. This means that we have $\mathit{MID}(\mathtt{proj}(P)) \leq c' \cdot \mathit{MID}(P) + c'' = c' \cdot 4 + c''$ for some $c', c'' \in \mathbb{N}$. Let $c = c' \cdot 4 + c''$. Suppose that k is much greater than c. This supposition requires that the number of basic instructions is not bounded. If the use of auxiliary basic instructions (such as basic instructions working on auxiliary Boolean cells) is allowed, then there are at most 2^c different basic instructions reachable in c steps. Let $i \in [1, 2^k]$. Then, in $\mathtt{proj}(P)$, for each $j \in [1, 2^k]$, some occurrence of a'_j is reachable from each occurrence of a_i without intermediate occurrences of a_i and $a'_1, \ldots, a'_{2^k}$. From one occurrence of a_i, at most 2^c basic instructions are reachable, but there are 2^k different instructions to reach. Therefore, there must be at least $2^k/2^c = 2^{k-c}$ different occurrences of a_i in $\mathtt{proj}(P)$. Consequently, the length of $\mathtt{proj}(P)$ is at least $2^k \cdot 2^{k-c} = 2^{2 \cdot k - c}$ instructions. This is a quadratic increase of the length, because the length of P is $12 \cdot 2^k + 4$ instructions. $\square$

We conclude from Theorem 1 that we are faced with super-linear increases of maximal internal delays if we strive for acceptable increases of program lengths on elimination of indirect jump instructions. In other words, indirect jump instructions are needed for instruction sequence performance. Semantically, we can eliminate indirect jump instructions by means of a projection, but we meet here two challenges of projectionism: explosion of size and degradation of performance.

5 Conclusions

We have shown that, in the case where the number of instructions is not bounded, there exist instruction sequences with direct and indirect jump instructions from which elimination of indirect jump instructions is possible without a super-linear increase of their maximal internal delay on execution only at the cost of a super-linear increase of their length. It is an open problem whether this result goes through in the case where the number of instructions is bounded.

Instruction sequences with direct jump instructions, indirect jump instructions and register transfer instructions are as expressive as instruction sequences with direct jump instructions and indirect jump instructions without register transfer instructions. We surmise that a projection that eliminates the register transfer instructions yields a result comparable to Theorem 1. However, we have not yet been able to provide a proof for that case. On the face of it, a proof for that case is much more difficult than the proof for the case treated in this paper.

For completeness, we mention that, in the line of research to which the work presented in this paper belongs, work that is mainly concerned with direct jump instructions includes the work presented in [3].

References

1. Bergstra, J.A., Loots, M.E.: Program algebra for sequential code. Journal of Logic and Algebraic Programming **51**(2), 125–156 (2002)
2. Bergstra, J.A., Middelburg, C.A.: Instruction sequences with indirect jumps. Scientific Annals of Computer Science **17**, 19–46 (2007)
3. Bergstra, J.A., Middelburg, C.A.: On the expressiveness of single-pass instruction sequences. Electronic Report PRG0813, Programming Research Group, University of Amsterdam (2008). Available from `http://www.science.uva.nl/research/prog/publications.html`. Also available from `http://arxiv.org/`: `arXiv:0810.1106v3 [cs.PL]`
4. Bergstra, J.A., Middelburg, C.A.: Instruction sequence notations with probabilistic instructions. Electronic Report PRG0906, Programming Research Group, University of Amsterdam (2009). Available from `http://www.science.uva.nl/research/prog/publications.html`. Also available from `http://arxiv.org/`: `arXiv:0906.3083v1 [cs.PL]`
5. Bergstra, J.A., van der Zwaag, M.B.: Mechanistic behavior of single-pass instruction sequences. `arXiv:0809.4635v1 [cs.PL]` at `http://arxiv.org/` (2008)
6. Gosling, J., Joy, B., Steele, G., Bracha, G.: The Java Language Specification, second edn. Addison-Wesley, Reading, MA (2000)
7. Hejlsberg, A., Wiltamuth, S., Golde, P.: C# Language Specification. Addison-Wesley, Reading, MA (2003)